Listing of Claims:

What is claimed is:

1. (Currently Amended) A method of compiling a program to be executed on a target microprocessor with multiple functional units of a same type, the method comprising opportunistically scheduling a redundant operation on one of the functional units that would otherwise be idle during a cycle and scheduling a comparison of results from the redundant operation.

2. (Cancelled)

3. (Original) The method of claim 2, further comprising:
causing a flag in the target microprocessor to be set when the comparison indicates an error.

4. (Original) The method of claim 2, further comprising:
setting a user-selectable level for an aggressiveness of said opportunistic scheduling.

5. (Original) A method of compiling a program to be executed on a target microprocessor, the method comprising:
identifying a cycle during which an operation is available for a first functional unit and no operation is available for a second functional unit, wherein the first and second functional units comprise functional units of a same type;
scheduling the operation for execution by both the first and second functional units during the cycle; and
scheduling a comparison of results obtained by the first and second functional units during a subsequent cycle.

6. (Original) The method of claim 5, wherein the first and second functional units comprise first and second floating point units of the target microprocessor.

7. (Original) The method of claim 5, wherein the first and second functional units comprise first and second arithmetic logic units of the target microprocessor.

8. (Original) The method of claim 5, wherein the results of the execution are stored in registers within the microprocessor, and the comparison of results compares contents of those registers.

9. (Original) The method of claim 5, wherein the target microprocessor includes at least three functional units of the same type.

10. (Original) The method of claim 9, further comprising:
identifying that during the cycle a second operation is available for a third functional unit of the same type and no operation is available for a fourth functional unit of the same type;
scheduling the second operation for execution by both the third and fourth functional units during the cycle; and
scheduling a comparison of results obtained by the third and fourth functional units during a subsequent cycle.

11. (Original) The method of claim 5, wherein the method is performed by a scheduler in a code generator of a program compiler.

12. (Original) The method of claim 11, wherein the program compiler comprises a native compiler for the target microprocessor.

13. (Original) The method of claim 11, wherein the program compiler comprises a cross compiler run on a different microprocessor.

14. (Original) The method of claim 5, further comprising:
causing a flag to be set when the comparison indicates an error.

15. (Original) The method of claim 14, further comprising:
if the error flag is set, then halting the execution and causing a notification to the user of the error flag.

16. (Currently Amended) A ~~program~~ computer-readable medium embedded with a compiler program for executing on a target microprocessor, the computer-readable medium comprising: ~~with~~
multiple equivalent functional units[[,]]; ~~the~~
a compiler comprising:
a code generator including a scheduler that opportunistically schedules a redundant operation on one of the functional units that would otherwise be idle during a cycle and schedules a comparison of results from the redundant operation.

17. (Cancelled)

18. (Currently Amended) A computer-readable ~~program product~~ medium embedded with a compiler program for execution on a target microprocessor, the computer-readable medium comprising: ~~with~~
multiple functional units of a same type[[,]]; ~~the program product comprising~~
executable code that includes a redundant operation scheduled for one of the functional units that would otherwise be idle during a cycle and also includes a subsequently scheduled comparison of results from the redundant operation for fault checking purposes.